TP Chapitre 9 : Les premiers algorithmes

Objectifs du TP

Objectifs du TP. À l'issue de cette séance, tu dois être capable de :

- décrire un algorithme en pseudo-code (entrée, traitement, sortie) avant de le traduire en Python ;
- parcourir séquentiellement un tableau avec une boucle `for` ;
- écrire les algorithmes du minimum et du maximum, en initialisant avec le premier élément ;
- additionner des valeurs (accumulateur) et calculer une moyenne ;
- écrire une recherche naïve : présence d'une valeur (booléen) et position de la première occurrence (-1 si absente) ;
- compter les valeurs qui vérifient un critère (compteur) ;
- écrire ces algorithmes dans des fonctions réutilisables et les tester avec des jeux de tests ;
- appliquer ces algorithmes à des objets en comparant un attribut (dans la partie C).

Consignes générales. Tous les exercices se traitent en Python (console ou fichier). Enregistre régulièrement ton travail. Comme dans le cours, commence par écrire le pseudo-code (*Entrée*, *Sortie*, *Algo*) avant de traduire en Python. Quand un programme ne fonctionne pas, lis le message d'erreur : il indique souvent la ligne et le type d'erreur. La partie B se fait *en autonomie* : tu écris des fonctions entières à partir d'une spécification, aucune ligne de code n'est fournie.

Partie A — Application guidée

Les exercices suivants se programment en Python. Ils sont classés du plus simple au plus difficile. Chaque exercice se termine par un *point de validation* (encadré vert) : le résultat exact que doit produire ton programme pour être juste. Complète les squelettes de code en respectant l'indentation.

Exercice A1 — Le catalogue de la boutique : prix le plus bas et prix le plus haut (minimum et maximum)

Une boutique en ligne affiche le prix de chacun de ses articles dans une liste. Le gérant veut repérer le prix le plus bas (pour préparer une promotion) et le prix le plus haut (pour l'assurance du magasin).

1. Écrire un programme qui affiche le prix le plus bas de la liste.

```
prix = [12.5, 8.9, 15.0, 6.4, 11.2, 9.8]
m = prix[0]
for p in prix:
    # A COMPLETER : si p est plus petit que m, m prend la valeur de p
print("Prix le plus bas :", m, "euros")
```

2. Écrire un programme qui affiche le prix le plus haut de la liste.

```
prix = [12.5, 8.9, 15.0, 6.4, 11.2, 9.8]
M = prix[0]
for p in prix:
    # A COMPLETER : si p est plus grand que M, M prend la valeur de p
print("Prix le plus haut :", M, "euros")
```

Point de validation

Le programme 1 doit afficher `Prix le plus bas : 6.4 euros` ; le programme 2, `Prix le plus haut : 15.0 euros`.

Exercice A2 — Le total et le prix moyen des achats du mois (somme et moyenne)

Sur ton relevé bancaire, tu veux connaître le total dépensé ce mois-ci et le prix moyen d'un achat.

```
depenses = [12.5, 30.0, 8.75, 42.3, 17.6]
somme = 0
for d in depenses:
    # A COMPLETER : ajouter d a la somme
moyenne = somme / len(depenses)
print("Total :", somme, "euros")
print("Prix moyen :", moyenne, "euros")
```

Point de validation

Le programme doit afficher `Total : 111.15 euros` et `Prix moyen : 22.23 euros` (un léger écart sur les décimales est normal : c'est la représentation des flottants).

Exercice A3 — Ce paiement figure-t-il dans mon relevé ? (recherche naïve)

Tu as payé 78,90 € dans un café. Avant de contester le débit, tu vérifies que ce paiement figure bien dans ton relevé bancaire du mois. Écrire un programme qui affiche `True` si la valeur cherchée est dans la liste, `False` sinon.

```
releve = [35.0, 12.5, 78.9, 42.0, 9.99]
valeur = 78.9
trouve = False
for montant in releve:
    # A COMPLETER : si montant == valeur, trouve prend la valeur True
print(trouve)
```

Point de validation

Le programme doit afficher `True`. Remplace `valeur` par `50.0` : il doit afficher `False`.

Exercice A4 — Combien d'achats dépassent 50 € ? (comptage)

Pour surveiller ton budget, tu veux savoir combien de tes achats du mois dépassent 50 €.

```
depenses = [12.5, 80.0, 45.0, 120.0, 33.0, 58.5]
seuil = 50
compteur = 0
for d in depenses:
    # A COMPLETER : si d depasse le seuil, augmenter compteur de 1
print("Achats de plus de", seuil, "euros :", compteur)
```

Point de validation

Le programme doit afficher `Achats de plus de 50 euros : 3` (les achats à 80, 120 et 58,5 € dépassent le seuil).

Partie B — En autonomie

Tu écris maintenant des fonctions *entières*, sans squelette. Pour chaque exercice : lis la spécification (entrées, traitement, sorties, contraintes), écris d'abord le pseudo-code sur papier (comme dans le cours), réponds aux questions de conception de l'encadré orange, puis écris ta fonction dans un fichier et teste-la avec les valeurs de l'encadré vert. L'objectif est de concevoir toi-même la solution : ne demande pas de lignes de code toutes faites.

Exercice B1 — Le meilleur jour de la caisse (fonction maximum)

Une petite caisse note chaque soir le solde de la journée : positif = bénéfice, négatif = perte. Le gérant veut connaître le meilleur jour de la semaine, c'est-à-dire le solde le plus élevé.

Énoncé. Écrire une fonction `meilleur_jour(soldes)` qui prend en entrée une liste de soldes quotidiens et qui renvoie le plus grand solde de la liste.

Spécification.

— *Entrées* : une liste `soldes` de flottants (soldes en euros, positifs ou négatifs), non vide.

— *Traitement* : parcourir la liste ; mémoriser dans `M` le plus grand solde rencontré, en initialisant avec `soldes[0]` et en mettant à jour quand un solde est plus grand.

— *Sorties* : le plus grand solde de la liste (`return`).

— *Contraintes* : fonction avec `return` ; initialisation avec le premier élément ; un seul parcours de la liste.

Pour t'aider — questions de conception (sans donner le code)

— Pourquoi initialiser `M` avec `soldes[0]` plutôt qu'avec `0` ? (pense à une semaine où tous les soldes sont négatifs : des jours de perte)

— Quelle comparaison permet de mettre `M` à jour : `>` ou `<` ?

— Que renvoie la fonction si l'on oublie le `return` ?

— Quel jeu de tests permet de vérifier que l'initialisation à `0` aurait donné un résultat faux ?

Point de validation

`meilleur_jour([25.5, 47.0, 12.9, 63.2, 31.0])` doit renvoyer `63.2` ; `meilleur_jour([-5.0, -12.0, -3.5])` doit renvoyer `-3.5` (et pas `0`).

Exercice B2 — Retrouver une dépense dans le relevé (fonction position)

Ton relevé bancaire liste les opérations du mois. Tu veux savoir à quelle position (quel indice) se trouve une dépense précise, pour la montrer à ton banquier.

Énoncé. Écrire une fonction `position_depense(releve, montant)` qui renvoie l'indice de la *première* occurrence de `montant` dans `releve`, ou `-1` si `montant` n'y figure pas.

Spécification.

— *Entrées* : une liste `releve` de flottants, et un montant (flottant).

— *Traitement* : parcourir les indices de 0 à `len(releve) - 1` ; dès que `releve[i] == montant`, renvoyer `i` ; si la boucle se termine sans succès, renvoyer `-1`.

— *Sorties* : un entier (indice de la première occurrence, ou `-1`).

— *Contraintes* : boucle `for` par indices (`range(len(releve))`) ; `return` immédiat dès que l'on trouve ; la méthode `index` est interdite.

Pour t'aider — questions de conception (sans donner le code)

— Pourquoi faut-il parcourir par indices plutôt que par valeurs ?

— Que renvoie la fonction si `montant` apparaît plusieurs fois ? Pourquoi ?

— Pourquoi choisir `-1` pour signaler « absent » ?

— Écris le pseudo-code de cette fonction avant de coder.

Point de validation

`position_depense([12.5, 78.9, 42.0, 78.9, 9.99], 78.9)` doit renvoyer `1` (première occurrence) ; avec `100.0`, elle doit renvoyer `-1`.

Exercice B3 — Le prix moyen du panier (fonction moyenne)

Pour comparer tes courses d'un magasin à l'autre, tu calcules le prix moyen des articles de ton panier.

Énoncé. Écrire une fonction `prix_moyen(panier)` qui prend en entrée une liste de prix et qui renvoie le prix moyen d'un article (la somme divisée par le nombre d'articles).

Spécification.

— *Entrées* : une liste `panier` de flottants (prix en euros), non vide.

— *Traitement* : additionner tous les prix avec un accumulateur, puis diviser par `len(panier)`.

— *Sorties* : le prix moyen (flottant, `return`).

— *Contraintes* : fonction ; une seule boucle ; la fonction `sum` est interdite (on réécrit l'algorithme du cours).

Pour t'aider — questions de conception (sans donner le code)

— Où doit être initialisé l'accumulateur : avant ou dans la boucle ? Pourquoi ?

— Pourquoi diviser par `len(panier)` et pas par une valeur fixe ?

— Que se passe-t-il si `panier` est vide ? (formule une *précondition*)

— Comment vérifier le résultat à la main avec le panier de test ?

Point de validation

`prix_moyen([3.5, 12.0, 7.25, 20.0, 8.4])` doit renvoyer environ `10.23`.

Exercice B4 — Combien de dépenses ce mois-ci ? (fonction de comptage)

Sur ton relevé, chaque opération est un nombre signé : positif = recette, négatif = dépense. Tu veux savoir combien de dépenses tu as faites ce mois-ci.

Énoncé. Écrire une fonction `compte_depenses(operations)` qui renvoie le nombre de montants *strictement négatifs* de la liste `operations`.

Spécification.

— *Entrées* : une liste `operations` de flottants signés (recettes et dépenses).

— *Traitement* : parcourir la liste ; pour chaque montant strictement négatif, augmenter un compteur de 1.

— *Sorties* : le nombre de dépenses (entier, `return`).

— *Contraintes* : fonction ; compteur initialisé à `0` avant la boucle ; un montant négatif représente une dépense.

Pour t'aider — questions de conception (sans donner le code)

— Quelle est la valeur initiale du compteur ? Pourquoi ?
— Comment reconnaître une dépense sur le relevé ?
— Que renvoie la fonction s'il n'y a aucune dépense ? Est-ce un résultat correct ?
— Écris le pseudo-code de cette fonction avant de coder.

Point de validation

`compte_depenses([120.0, -45.5, 200.0, -30.0, -12.75, 55.0])` doit renvoyer `3` ; `compte_depenses([50.0, 25.0])` doit renvoyer `0`.

Partie C — Exercice de réflexion

À finir à la maison

Mon analyseur de portefeuille d'actions.

Contexte. Tu suis le cours d'une action (ou d'une cryptomonnaie) pendant un mois. Tu veux un programme qui analyse la liste des cours quotidiens et affiche un bilan : le cours le plus bas (bon moment pour acheter), le cours le plus haut (bon moment pour vendre), le jour où chacun a eu lieu, le cours moyen sur la période, le nombre de jours de hausse et le gain potentiel (le plus haut moins le plus bas).

Objectif. Écrire un programme qui prend une liste de cours quotidiens (en euros) et affiche ce bilan complet. Tu es libre d'organiser ton programme comme tu veux : script simple ou fonctions.

Questions à se poser avant de coder.

— Quelles sont les données d'entrée ? (la liste des cours quotidiens)
— Quels algorithmes du chapitre vais-je réutiliser ? (minimum, maximum, moyenne, position, comptage)
— Faut-il parcourir par valeurs ou par indices ? Dans quelles parties du bilan ?
— Comment découper le problème en fonctions ?
— Comment vérifier que le programme est juste ? (choisir un jeu de données simple dont on connaît le résultat à la main)

Étapes suggérées.

1. Version 0 : liste fixe de 5 cours connus, par exemple `[48.5, 52.0, 47.2, 55.5, 51.0]` ; afficher le minimum, le maximum et la moyenne ; vérifier les résultats à la main.
2. Version 1 : ajouter la position (le jour) du minimum et du maximum — le premier cours correspond au jour 1.
3. Version 2 : compter les jours de hausse (le cours du jour est supérieur à celui de la veille) et les jours de baisse.
4. Version 3 : afficher un bilan complet et bien présenté, avec le gain potentiel (maximum − minimum).
5. Version 4 (bonus) : générer 30 cours aléatoires entre 10 et 60 € (`import random as rd` ; `rd.uniform(10, 60)`) et relancer le bilan.

6. Bonus objets : définir une classe `Action` avec un nom et un cours ; créer une liste d'actions ; afficher l'action la moins chère et l'action la plus chère en comparant l'attribut `cours` (comme la classe `Cadeau` du cours).

Contraintes. Données crédibles (cours entre 10 et 60 €) ; programme testé avec une liste dont tu connais les résultats ; programme commenté ; les messages peuvent être écrits sans accents si ton éditeur pose problème.

Fin du TP — la partie C est à finir à la maison.